

# Critique of J. Kim's " $\mathbf{P}$ is not equal to $\mathbf{NP}$ by Modus Tollens"

Dan Hassin, Adam Scrivener, and Yibo Zhou

Department of Computer Science  
University of Rochester  
Rochester, NY 14627, USA

April 29, 2014

## Abstract

This paper is a critique of version three of Joonmo Kim's paper entitled " $\mathbf{P} \neq \mathbf{NP}$  by Modus Tollens." After summarizing Kim's proof, we note that the logic that Kim uses is inconsistent, which provides evidence that the proof is invalid. To show this, we will consider two reasonable interpretations of Kim's definitions, and show that " $\mathbf{P} \neq \mathbf{NP}$ " does not seem to follow in an obvious way using any of them.

## 1 Introduction

The abstract of Kim's paper [2] is as follows:

An artificially designed Turing Machine algorithm  $\mathbf{M}^o$  generates the instances of the satisfiability problem, and check their satisfiability. Under the assumption  $\mathbf{P} = \mathbf{NP}$ , we show that  $\mathbf{M}^o$  has a certain property, which, without the assumption,  $\mathbf{M}^o$  does not have. This leads to  $\mathbf{P} \neq \mathbf{NP}$  by modus tollens.

In this paper we will critique Kim's proof that  $\mathbf{P} \neq \mathbf{NP}$ , a proof that claims to solve the famous  $\mathbf{P}$  vs.  $\mathbf{NP}$  problem, widely agreed to be the most important unsolved problem in computer science. We will begin by outlining Kim's paper in detail. Then we will examine several problems with his paper, including the many ambiguous definitions, wordings, and explanations throughout his paper. We will address both of the possible interpretations of his most ambiguous definition, and describe how either interpretation arrives at a contradiction. Finally, we provide a comment on the invalidity of the final commentary of Kim's paper.

## 2 Kim's argument

Most of the arguments in Kim's paper are not rigorous, some constructions are not shown, his definitions are not precise, and we believe his paper contains many notational mistakes. For this reason, this section will restate his paper in detail as we understand it so that it may be refuted rigorously.

## 2.1 Cook-Levin theorem

By the Cook-Levin theorem, we can construct a series of Boolean clauses based on the action of any arbitrary Turing machine on a given input  $x$ , such that the series of clauses is satisfiable if and only if it accepts  $x$ . Together, Kim calls this unit of clauses  $\mathbf{c}$ . In the proof of this theorem given by Garey and Johnson, these Boolean clauses in  $\mathbf{c}$  are grouped into  $G_1, G_2, G_3, G_4, G_5$ , and  $G_6$  by the different parts of the computation that they enforce. One group of particular interest to Kim is  $G_4$ , which enforces that “at time 0, the computation is in the initial configuration of its checking stage for input  $x$ ” [1]. The rest of the groups of clauses,  $G_1, G_2, G_3, G_5$ , and  $G_6$ , are each concerned with the run of the machine, its transitions, and that it ends at a final state [1].

For a given  $\mathbf{c}$ , he lets  $\mathbf{c}^x$ , what he calls the “input-part,” be  $G_4$ , the group that is concerned with asserting the initial configuration. He lets  $\mathbf{c}^r$ , the “run-part,” be the rest of the clauses, those concerned with the run of the machine.

## 2.2 Construction of $\mathbf{M}$

**Definition:** An *accepting computation* of a Turing machine  $T$  on input  $y$  is a finite sequence of configurations and transitions of  $T$  (starting at the initial state of  $T$  and with  $y$  on its tape) that ends in an accepting state of  $T$ .

Kim proposes a Turing machine algorithm  $\mathbf{M}$ , which has in its code a finite list  $\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_n$  of Boolean formulas, each which have been encoded from arbitrary accepting computations from arbitrary Turing machines, using the Cook-Levin construction described above. For each  $\mathbf{c}_j$  in this list, we can strip its “input-part” to obtain the list of “run-parts”  $\mathbf{c}_1^r, \mathbf{c}_2^r, \dots, \mathbf{c}_n^r$ . Call this list of “run-parts” from arbitrary machines  $C_{\mathbf{M}}^r$ .

Kim notes that we can construct countably many  $\mathbf{M}$ ’s,  $\mathbf{M}_1, \mathbf{M}_2, \mathbf{M}_3, \dots, \mathbf{M}_i$ , where each  $\mathbf{M}_i$  includes in its program code the list of Boolean formulas  $C_{\mathbf{M}_i}^r$ , as described above. Kim defines the behavior of a given  $\mathbf{M}_i$  as follows.

$\mathbf{M}_i$  = “On input  $y$ :

1. Using the Cook-Levin construction, compute the Boolean clause  $\mathbf{c}^y$  (i.e.,  $G_4$ , the clause that enforces the initial machine state of  $\mathbf{M}_i$  and the placement of input  $y$  on its tape.)<sup>1</sup>
2. For each  $\mathbf{c}_j^r \in C_{\mathbf{M}_i}^r$ :
  - (a) Concatenate  $\mathbf{c}_j^r$  with  $\mathbf{c}^y$  to form  $\mathbf{c}_j$ .
  - (b) Give  $\mathbf{c}_j$  to a *SAT*-solver module, and increment a counter if it returns that  $\mathbf{c}_j$  is satisfiable (i.e., if it is a valid accepting computation with input  $y$ .)
3. Accept if the number accumulated by the counter is odd.”

Let  $C_{\mathbf{M}_i, y}$  be the list of each  $\mathbf{c}$  that appears in the run of  $\mathbf{M}_i$  on input  $y$ . More formally,  $C_{\mathbf{M}_i, y} = \mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_n$  such that  $\mathbf{c}_j = (\mathbf{c}^y \wedge \mathbf{c}_j^r)$ , where  $\mathbf{c}_j^r \in C_{\mathbf{M}_i}^r$ .

## 2.3 Defining $\mathbf{M}^o$

Kim introduces the idea of a *particular transition table* of a Turing machine, which he says is a transition table that “may produce an accepting computation

by running on a Turing Machine” [2]. He observes that “each of all accepting computations may have its particular transition table, i.e., the table can be built by collecting all the distinguished transitions from the computation, where we know that a computation is a sequence of the transitions of configurations of a Turing Machine” [2].

Kim then proposes the machine  $\mathbf{M}^o \in \{\mathbf{M}_1, \mathbf{M}_2, \dots, \mathbf{M}_i\}$  such that, for some input  $y$ , there exists a  $\mathbf{c}^o \in C_{\mathbf{M}^o, y}$  that describes an accepting computation on input  $y$  (call this accepting computation  $\widehat{ac}_{\mathbf{c}^o}$ ) for which there exists a particular transition table  $\mathbf{t}$  which is also a particular transition table for the accepting computation of the run of  $\mathbf{M}^o$  on  $y$  (call this  $\widehat{ac}_{\mathbf{M}^o}$ .) Note, rather importantly, that  $\widehat{ac}_{\mathbf{c}^o}$  and  $\widehat{ac}_{\mathbf{M}^o}$  are both accepting computations with respect to input  $y$ .

## 2.4 “ $\mathbf{P} \neq \mathbf{NP}$ ”

**Definition:** [2] A particular transition table  $\mathbf{t}$  is  $D_{sat}$  if it “deterministically describes  $\mathbf{M}^o$ ’s transitions and the SAT-solver module in  $\mathbf{M}^o$  runs deterministically in a poly-time for the length of  $\mathbf{c}$ .”

Kim’s proof of  $\mathbf{P} \neq \mathbf{NP}$  is as follows.

$P_1$ :  $\mathbf{P} = \mathbf{NP}$ ,

$P_2$ :  $\mathbf{M}^o$  exists,

$P_3$ : there exists  $\mathbf{t}$ , which is  $D_{sat}$ .

By modus tollens,  $(P_1 \Rightarrow (P_2 \Rightarrow P_3)) \wedge (\neg(P_2 \Rightarrow P_3))$  may conclude  $\neg P_1$ . [2]

He argues that  $P_1 \Rightarrow (P_2 \Rightarrow P_3)$  because, if such an  $\mathbf{M}^o$  exists, by definition a  $\mathbf{t}$  must exist, and that  $\mathbf{t}$  is  $D_{sat}$ , because if  $\mathbf{P} = \mathbf{NP}$ , the SAT-solver module (which is known to be  $\mathbf{NP}$ -complete) would run in deterministic polynomial time.

This argument is sufficient to show that  $P_1 \Rightarrow (P_2 \Rightarrow P_3)$ . All Kim must show now is that  $P_2 \Rightarrow P_3$  results in a contradiction, thus showing  $\neg(P_2 \Rightarrow P_3)$ , and proving by modus tollens  $\neg P_1$ . But first, he argues (unnecessarily) that  $P_2$  is true. And in fact, he is actually arguing that  $P_2 \wedge P_3$  is true. The following argument [2], while irrelevant, is shown for completeness.

We can show that  $P_2$  is true, as follows. For any chosen  $\mathbf{c}^o$ , build two non-deterministic particular transition tables for  $\widehat{ac}_{\mathbf{M}^o}$  and  $\widehat{ac}_{\mathbf{c}^o}$  separately, and then merge the two so that one of the two computations can be chosen selectively from the starting state during the run.  $\mathbf{M}^o$  may exist by this  $\mathbf{t}$ , which is  $ND_{sat}$ .

Kim provides very little to explain precisely what this “merging process” of tables is. Below is a construction of what we assume his merging process to be. Consider transition tables  $\delta_M$  and  $\delta_{M'}$  for machines  $M$  and  $M'$  respectively: (For brevity, head movements and writes to the tape are omitted; only state transitions are shown.)

$\delta_M$	a	b	...
$q_0$	$q_w$	$q_x$	
$q_1$	$q_y$	$q_z$	
$\vdots$			$\ddots$

$\delta_{M'}$	a'	b'	...
$q'_0$	$q'_w$	$q'_x$	
$q'_1$	$q'_y$	$q'_z$	
$\vdots$			$\ddots$

Using  $\delta_M$  and  $\delta_{M'}$ , we can produce the non-deterministic particular transition table  $\delta_{M,M'}$  that “describes,” or that can “produce” both  $\hat{ac}_M$  and  $\hat{ac}_{M'}$ , as generated by  $\delta_{M,M'}$ , for arbitrary input  $y$ :

$\delta_{M,M'}$	$\epsilon$	a	b	...	a'	b'	...
$q_{start}$	$\{q_0, q'_0\}$	$\emptyset$	$\emptyset$		$\emptyset$	$\emptyset$	
$q_0$	$\emptyset$	$\{q_w\}$	$\{q_x\}$		$\emptyset$	$\emptyset$	
$q_1$	$\emptyset$	$\{q_y\}$	$\{q_z\}$		$\emptyset$	$\emptyset$	
$\vdots$				$\ddots$			
$q'_0$	$\emptyset$	$\emptyset$	$\emptyset$		$\{q'_w\}$	$\{q'_x\}$	
$q'_1$	$\emptyset$	$\emptyset$	$\emptyset$		$\{q'_y\}$	$\{q'_z\}$	
$\vdots$							$\ddots$

## 2.5 Contradiction argument

Kim’s proof by contradiction to prove  $\neg(P_2 \Rightarrow P_3)$  is as follows. By way of contradiction, he assumes  $(P_2 \Rightarrow P_3)$  to be true, i.e., that “if  $\mathbf{M}^o$  exists then there exists  $\mathbf{t}$ , which is a  $D_{sat}$  particular transition table for both  $\hat{ac}_{\mathbf{M}^o}$  and  $\hat{ac}_{\mathbf{c}^o}$ ” [2]. He then claims that, since the same transition table  $\mathbf{t}$  can generate both  $\hat{ac}_{\mathbf{M}^o}$  and  $\hat{ac}_{\mathbf{c}^o}$ , which share the same input  $y$ , “it is concluded that both  $\hat{ac}_{\mathbf{M}^o}$  and  $\hat{ac}_{\mathbf{c}^o}$  are exactly the same computation, i.e., all the transitions of the configurations of  $\hat{ac}_{\mathbf{M}^o}$  and those of  $\hat{ac}_{\mathbf{c}^o}$  are exactly the same” [2].

Now, he lets  $i$  be the number of transitions between configurations in  $\hat{ac}_{\mathbf{M}^o}$ ,  $j$  the number of clauses of  $\mathbf{c}^o$ , and  $k$  the number of transitions between configurations in  $\hat{ac}_{\mathbf{c}^o}$ .

He argues that during the run of  $\mathbf{M}^o$  on input  $y$ , all the clauses of  $\mathbf{c}^o$  will have to be loaded on the tape of  $\mathbf{M}^o$ , as well as the clauses of all other  $\mathbf{c}$ ’s  $\in C_{\mathbf{M}^o, y}$ , so  $i > j$ . And, since each transition of an accepting computation is described by more than one clause [1], we conclude  $j > k$ , and thus  $i > j > k$ .

However, Kim argues that a contradiction arises here. The previous conclusion that  $\hat{ac}_{\mathbf{M}^o}$  and  $\hat{ac}_{\mathbf{c}^o}$  are exactly the same computation would imply that  $i = k$ , which contradicts  $i > j > k$ . Thus, he claims  $\neg(P_2 \Rightarrow P_3)$ .

## 3 Critique

During our analysis of his argument, we identified several flaws in Kim’s proof which we critique here in detail.

### 3.1 Invalidity of logical argument

Kim’s argument centers around the definition of  $D_{sat}$ , as well as this fact: if  $\mathbf{P} = \mathbf{NP}$  then the particular transition table that is implied by  $\mathbf{M}^o$ ’s existence is  $D_{sat}$ . Kim then attempts to arrive at a contradiction by showing that such a

particular transition table cannot exist. However, in his proof by contradiction, he does not use the fact that  $\mathbf{t}$  is  $D_{sat}$ , so the assumption (that if there exists an  $\mathbf{M}^o$  then there exists a  $\mathbf{t}$  that is  $D_{sat}$ ) is equivalent to  $(\mathbf{M}^o \text{ exists}) \Rightarrow (\mathbf{t} \text{ exists})$ . Note that, by definition,  $\mathbf{M}^o$  exists if and only if  $\mathbf{t}$  exists. Therefore, Kim cannot possibly prove that  $(\mathbf{M}^o \text{ exists}) \Rightarrow (\mathbf{t} \text{ exists})$  is false. This fact provides evidence that his proof must be invalid, which we will presently show.

### 3.2 Ambiguities with accepting computations and particular transition tables

An error arises in Kim's final contradiction that  $P_2$  does not imply  $P_3$ , namely that since  $\mathbf{M}^o$  exists, a  $D_{sat}$  particular transition table of both  $\hat{ac}_{\mathbf{c}^o}$  and  $\hat{ac}_{\mathbf{M}^o}$  exists. Kim argues that the existence of this particular transition table implies that  $\hat{ac}_{\mathbf{c}^o}$  and  $\hat{ac}_{\mathbf{M}^o}$  are equivalent accepting computations.

Here, Kim's definition of an *accepting computation* is of crucial importance. Michael Sipser [3] offers the following definition of an *accepting computation history*:

Let  $M$  be a Turing machine and  $w$  an input string. An *accepting computation history* for  $M$  on  $w$  is a sequence of configurations,  $C_1, C_2, \dots, C_l$ , where  $C_1$  is the start configuration of  $M$  on  $w$ ,  $C_l$  is an accepting configuration of  $M$ , and each  $C_i$  legally follows from  $C_{i-1}$  **according to the rules of  $M$** . (Emphasis added.)

Although Sipser refers to an accepting computation *history*, we infer from Kim's own paper that this definition is equivalent to simply *accepting computation*: "...we know that a computation is a **sequence of the transitions of configurations** of a Turing Machine" (emphasis added) [2].

Note that Sipser's definition suggests, as would common intuition, that an accepting computation relies on the transition table of the given machine running it. However, Kim is vague in describing how particular transition tables and accepting computations relate. One could interpret it in one of two ways. Either,

1. An accepting computation is produced by a given Turing machine and its own transition table.
2. An accepting computation can be produced by a given particular transition table, not necessarily that of the original machine, that can describe each transition between configurations.

We believe that an error arises when Kim operates under the first interpretation for his claim that  $i > j > k$ , and the second for the  $i = k$  claim. To produce a consistent and coherent proof, the paper can only operate under one interpretation. In the following sections, we will address both interpretations independently and show that under either one, his contradiction is invalid.

#### 3.2.1 First interpretation

By this interpretation,  $\hat{ac}_{\mathbf{M}^o}$  and  $\hat{ac}_{\mathbf{c}^o}$  are accepting computations from different Turing machines entirely, which behave in very different ways.  $\mathbf{M}^o$ , on input  $y$ , concatenates its own  $\mathbf{c}^y$  with each of the Boolean formulas in  $C_{\mathbf{M}^o}^r$ , then runs

a *SAT*-solver module on each  $\mathbf{c}$ , counting the  $\mathbf{c}$ 's that are accepted. On the other hand, each  $\hat{ac}_{\mathbf{c}}$  is just an arbitrary accepting computation of some Turing machine  $\mathbf{M}$  on an input  $y$ . Under this interpretation, it is not obvious that any  $\hat{ac}_{\mathbf{M}^o}$  is equivalent to any  $\hat{ac}_{\mathbf{c}^o}$ . The argument that Kim gives as proof that some  $\hat{ac}_{\mathbf{M}^o}$  is equivalent to some  $\hat{ac}_{\mathbf{c}^o}$  is that one can create a particular transition table that is a transition table for both  $\hat{ac}_{\mathbf{M}^o}$  and  $\hat{ac}_{\mathbf{c}^o}$ . However, the “merging” technique that Kim uses to show that any  $\mathbf{t}$  can be made from two transition tables can be shown to be invalid.

This technique produces a new transition table, which contains new states (as it must include the set of states from both machines) and possibly new alphabet characters. Thus, it cannot be said that the new particular transition table is the same transition function as either original machine, or even a “compatible” one, since it operates on a set of states that is different from the machine’s original set of states, and would thereby be malformed.

Therefore, under this interpretation, Kim’s argument that  $i = k$  follows from there existing some  $\mathbf{t}$  which can produce  $\hat{ac}_{\mathbf{M}^o}$  and  $\hat{ac}_{\mathbf{c}^o}$  is invalid, since  $\hat{ac}_{\mathbf{M}^o}$  and  $\hat{ac}_{\mathbf{c}^o}$  are computations produced by transition tables necessarily different from their original machines, and thus  $i > j > k$  is correct, and there is no inconsistency.

### 3.2.2 Second interpretation

In the second interpretation, we will assume that accepting computations can be produced by the particular transition table  $\mathbf{t}$ . Then we may conclude that since  $\mathbf{t}$  is a particular transition table for  $\hat{ac}_{\mathbf{M}^o}$  and  $\hat{ac}_{\mathbf{c}^o}$ , then  $\hat{ac}_{\mathbf{M}^o}$  and  $\hat{ac}_{\mathbf{c}^o}$  are equivalent as **accepting computations produced by a particular transition table for some input  $y$** . Note that these accepting computations are not necessarily the same as the accepting computations produced by their respective Turing machines’ transition tables. So, when Kim concludes that the number of transitions in  $\hat{ac}_{\mathbf{M}^o}$  must be larger than the number of transitions in  $\hat{ac}_{\mathbf{c}^o}$  as a contradiction, he is no longer comparing the same accepting computations, so that fact is not contradictory. Given the nature of  $\mathbf{M}^o$  and  $\mathbf{c}^o$ ,  $i > j > k$  does not follow, because  $\hat{ac}_{\mathbf{M}^o}$  and  $\hat{ac}_{\mathbf{c}^o}$  are indeed the same computations produced by  $\mathbf{t}$ , and analysis based on their original respective machines does not apply.

### 3.3 Comment on Kim’s “commentary”

At the end of his paper, Kim verifies that his given proof could not also be used to prove that  $\mathbf{P} = \mathbf{NP}$ . This verification is very brief and relies heavily on the assumption that the proof that his paper presents makes accurate assumptions and logical inferences. It is essentially a retelling of his argument with reversed assumptions and conclusions. Clearly, assuming that his original proof is correct, it can be used to refute the possibility of proving the opposite statement, but it does not tell us anything about the validity of the original proof.

## 4 Conclusion

From our interpretation of Kim’s paper, the main problems stem from a severe lack of rigor, numerous misunderstandings, and occasional inconsistencies in his

definitions. In his main argument, he derives a contradiction from the properties of a Turing machine and the  $D_{sat}$  property of a particular transition table. In our main argument, we point out that there is an inconsistency here that renders the main proposition of his supposed contradiction invalid in the two possible interpretations of his definition regarding *accepting computation*.

## Acknowledgments

We thank Lane A. Hemaspaandra and Joe Izraelevitz for helpful comments on a preliminary draft of this critique. All claims, opinions, and errors in the present, substantially revised critique are the sole responsibility of the authors.

## Notes

<sup>1</sup>The Garey and Johnson construction that Kim uses for the *SAT* reduction, even for group  $G_4$ , is reliant on a machine so that its initial state can be encoded: “at time 0, the computation is in the initial configuration of its checking stage for input  $x$ ” [1].

However, Kim does not specify which machine to use in this construction of  $\mathbf{c}^y$ . So that the machine code for  $\mathbf{M}_i$  is well defined, we use  $\mathbf{M}_i$  itself as the machine used in the  $\mathbf{c}^y$  construction, but the following argument will show that in fact the choice of machine does not matter. (And since Garey and Johnson’s construction is not the only conceivable construction that maps machine actions to clauses (with separate “input” and “run” parts) as per the Cook-Levin theorem, our proof is independent of the details of that specific construction.)

Suppose that a given Boolean formula  $\mathbf{c}^y$  was constructed from machine  $M$  on input  $y$ , and that an arbitrary Boolean formula  $\mathbf{c}^r$  was constructed from machine  $M^r$ .

- If the only relevant information in  $\mathbf{c}^y$ , even being constructed from  $M$  specifically, is the placement of  $y$  on the tape (i.e., that the construction encodes all initial states for all machines the same way), then indeed concatenating  $\mathbf{c}^r$  from  $M^r$  will produce the same  $\mathbf{c}$  that the construction would produce for  $M^r$  on input  $y$ .
- If the information in  $\mathbf{c}^y$  uses specific elements of  $M$ , like  $M$ ’s start state, it is still possible that the clauses of  $\mathbf{c}^r$  are “compatible” with  $\mathbf{c}^y$  (i.e., the construction produces the same  $\mathbf{c}^y$ ’s for both  $M$  on input  $y$  and  $M^r$  on input  $y$ ), and so it is *possible* that  $(\mathbf{c}^y \wedge \mathbf{c}^r)$  is satisfiable.
- If the information in  $\mathbf{c}^y$  is absolutely specific to  $M$  and is entirely “incompatible” with any other machine Boolean encodings, then no  $\mathbf{c}$  will be satisfiable, meaning that no  $\hat{a}\hat{c}_{\mathbf{c}}$  and thus no  $\mathbf{c}^o$  (explained in Section 2.3) can exist. This would make the antecedent in  $(P_2 \Rightarrow P_3)$  in Section 2.4 false, which would make the statement true, and thus the logical model is valid regardless of the truth value of  $P_1$  ( $\mathbf{P} = \mathbf{NP}$ .)

## References

- [1] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman & Co., New York, USA, 1979.
- [2] Joonmo Kim. P is not equal to NP by Modus Tollens. *arXiv.org*, CoRR, 2014. <http://arxiv.org/abs/1403.4143v3>.
- [3] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, Boston, MA, USA, third edition, 2013.